**UV Adoption Guide for Python Projects**

*A simple and clear guide written for any reader*

## How to Use This Document

This guide explains what uv is, why it can help, and how to use it in your projects. It is written in simple language. It includes a Jira spike story that you can copy. It also has steps to create a new project, steps to move an old project from pip to uv, a Docker example, and a list of pros and cons. There is also a short help section at the end.

This guide does not include any steps for continuous integration or continuous delivery.

# 1. What Is uv

uv is a fast tool for Python that installs and manages packages. It is made by the team that also made Ruff and Black. It is written in Rust so it is very fast. You can think of uv as a modern replacement for pip. It can also make and manage your virtual environment.

The most important idea is that uv can make a lock file named uv.lock. A lock file helps you install the exact same package versions on every computer. This gives you the same results every time.

## Key Points

• uv installs packages much faster than pip in most cases.

• uv can create and manage a virtual environment for you with the commands uv venv and uv sync.

• uv uses a lock file to make installs repeatable and stable.

• uv works with pyproject.toml. It can also work with requirements.txt if you still need it.

• You can run your code inside the managed environment with uv run.

## Common Commands

# Install uv one time on your machine  
pip install uv  
  
# Make a virtual environment and install based on the project and lock file  
uv venv  
uv sync  
  
# Add a package for your application  
uv add requests  
  
# Add a package only for development  
uv add --dev pytest  
  
# Run a script inside the project environment  
uv run main.py  
  
# Export pinned requirements from a locked project if some tool needs it  
uv export -o requirements.txt

# 2. Benefits of Using uv

• Speed. Installs are faster so development and builds take less time.

• Reproducible results. The lock file keeps versions the same for all users.

• Simple steps. One tool can make the environment and install packages.

• Works with modern project files. uv reads pyproject.toml and writes uv.lock.

• Easy to move from pip. You can start with your current files and improve later.

# 3. Jira Spike User Story

Title: Investigate the move from pip to uv for Python dependency management

Story: As a development team we want to learn if uv can replace pip in our projects so that we can improve speed, reproducibility, and developer experience.

## Acceptance Criteria

1. Write a short table and notes that compare pip and uv. Include speed, lock file use, and basic commands.

2. Show that uv works with our current steps. This includes requirements.txt installs, virtual environments, editable installs, and private package indexes if we have them.

3. Create a small prototype. Do it in two ways. Option A keeps requirements.txt and uses uv pip sync. Option B uses pyproject.toml and a lock file. Both must install and run tests.

4. Write a short guide for local setup for new developers.

5. List risks and problems with a simple plan to reduce each one.

6. Give a clear recommendation. Choose one of these: move fully to uv, use uv as a drop in speed up, or stay with pip. Explain why.

## Research Tasks

• Task 1. Install uv and create a small test project. Try uv add, uv lock, uv sync, and uv run.

• Task 2. Measure time to install with pip and with uv on a real project. Do it once with a cold cache and once again with a warm cache. Write down the times and list your system and Python version.

• Task 3. Try the locked project on two different systems if possible. Confirm that the environment is the same and that tests pass.

• Task 4. If you use a private package index test the install with a token or secret. Do not print secrets in logs.

• Task 5. Write the migration steps and a roll back plan. The roll back plan is to return to pip steps if something fails.

• Task 6. Summarize your findings and present a short plan for adoption.

# 4. Create a New Project With uv

Follow these steps to start a clean project.

mkdir cool\_app  
cd cool\_app  
  
# Create project files  
uv init  
  
# Add run time packages  
uv add fastapi uvicorn  
  
# Add development packages  
uv add --dev pytest pytest-cov  
  
# Make a lock file and install everything  
uv lock  
uv sync  
  
# Run a quick server example  
uv run python -c "print('hello from uv')"

The uv init command creates a pyproject.toml file. The uv lock command creates the uv.lock file. The uv sync command creates a .venv folder and installs all packages based on the lock file.

# 5. Migrate From an Existing Project That Uses pip

There are two safe paths. You can start with a drop in change that keeps your current files. Later you can move to the full uv project if you want a lock file and a simpler setup.

## Option A. Drop in change and keep requirements.txt

Use this when you want speed now and do not want to change file formats yet.

# One time  
pip install uv  
  
# In your project folder  
uv venv  
source .venv/bin/activate # Windows users can run .venv\Scripts\activate  
uv pip sync requirements.txt  
  
# If you used editable install before  
uv pip install -e .  
  
# Run tests to confirm  
pytest -q

This gives faster and more stable installs while your team still uses the same files.

## Option B. Full uv project with pyproject.toml and uv.lock

Use this when you want a modern setup with a lock file. This gives repeatable installs across computers.

# Create project file if missing  
uv init  
  
# Import your current requirements as input and keep pins with constraints  
# If you have both requirements.in and requirements.txt  
uv add -r requirements.in -c requirements.txt  
  
# If you also have dev requirements  
# Remove any -r includes inside the dev file to avoid duplicates  
uv add --dev -r requirements-dev.in -c requirements-dev.txt  
  
# Make the lock file and install  
uv lock  
uv sync  
  
# Now you can remove old steps that used pip install -r  
# If some tool still needs requirements.txt you can export it  
uv export -o requirements.txt

After this change your project is defined by pyproject.toml and uv.lock. Developers only need to run uv sync.

# 6. Private Package Index and Credentials

Many teams use a private package index. uv supports the same idea as pip. It can read a base index and extra index settings from the command line or from environment variables.

# Use a base index url for one command  
uv pip install --index-url=https://user:token@private.example.com/simple packageA  
  
# Use extra index url as well  
uv pip install --index-url=https://user:token@private.example.com/simple --extra-index-url=https://pypi.org/simple packageB  
  
# Prefer environment variables so tokens are not typed into history  
export UV\_INDEX\_URL="https://token@private.example.com/simple"  
uv pip sync requirements.txt

For shared systems place secrets in a secret store. Do not print them in logs.

# 7. Use Docker With uv and pyproject

Below is a simple and production ready Dockerfile that uses uv and the project files pyproject.toml and uv.lock. It does not use requirements.txt.

# syntax=docker/dockerfile:1.7  
  
FROM python:3.11-slim AS builder  
WORKDIR /app  
  
# Install uv  
RUN pip install --no-cache-dir uv  
  
# Copy only files that define dependencies first to improve cache hit rate  
COPY pyproject.toml uv.lock ./  
  
# Install only application packages for run time  
RUN --mount=type=cache,target=/root/.cache/uv uv sync --no-dev  
  
# Copy the rest of the source code  
COPY . .  
  
# Optional editable install if your app is a package  
RUN uv pip install -e .  
  
# Runtime stage  
FROM python:3.11-slim AS runtime  
WORKDIR /app  
  
# Copy the virtual environment and the code  
COPY --from=builder /app/.venv /app/.venv  
COPY --from=builder /app /app  
  
# Use the project virtual environment by default  
ENV PATH="/app/.venv/bin:$PATH" PYTHONDONTWRITEBYTECODE=1 PYTHONUNBUFFERED=1  
  
# Example command. Change to your real entry point  
CMD ["python", "main.py"]

This pattern keeps images small and fast to build. It also makes rebuilds faster because Docker can reuse layers when your code changes but your dependencies do not change.

# 8. Pros and Cons of uv

## Pros

• Faster installs which save time for developers and in container builds.

• Lock file gives repeatable results across computers and operating systems.

• Simple and modern project layout with pyproject.toml and a lock file.

• Works with existing projects and can use requirements.txt if needed.

• Clear commands that combine virtual environment setup and installs.

## Cons

• It is a newer tool. Some team members may not know it yet.

• Some rare tools still expect only pip commands. In those cases you can use the uv pip commands or export requirements from the lock file.

• If you have custom native builds you may still need system packages. This is not a uv issue but you should plan for it.

# 9. Best Practices

• Commit pyproject.toml and uv.lock to the repository.

• Use uv sync to install for both development and production to get the same versions.

• When you add or remove a package run uv lock then run uv sync again.

• Keep a short developer set up section in the README with uv commands.

• If you must provide requirements.txt to another system, export it from the lock file so it matches the project.

# 10. Troubleshooting and Frequently Asked Questions

Q: I get an import error after switching to uv. What should I do

A: Make sure you ran uv sync and that you are using the virtual environment. You can check by running which python on mac or linux or where python on Windows.

Q: I need to run a script with some extra packages for a one time task. Do I need to add them to the project

A: No. Use uv run with the with option. Example:

uv run --with 'rich>=13' scripts/demo.py

Q: I need a certain Python version. Can uv help with that

A: Yes. Use uv run with a python value. Example:

uv run --python 3.11 main.py

Q: Another system still needs a requirements.txt file. What do I do

A: Export a pinned file from your locked project so it matches versions exactly.

uv export -o requirements.txt

## Final Notes

uv is a strong choice for new and old Python projects. It is fast, simple, and gives stable results. You can start small by keeping your current files and using uv pip sync. Later you can move to a full uv project with a lock file. The Docker pattern in this guide will help you build smaller and faster images.